

# C Programming

## 함수

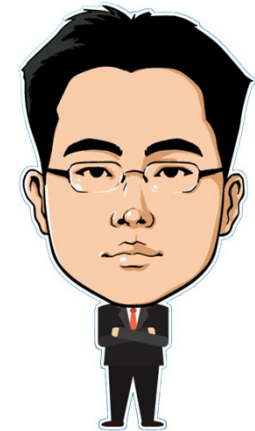
(Functions)



Seo, Doo-Ok

Clickseo.com

clickseo@gmail.com



# 목 차



백문이불여일타(百聞而不如一打)

- 함수의 이해
- 저장 공간 분류
- 함수와 포인터
- 재귀 함수



# 함수의 이해



- 함수의 이해

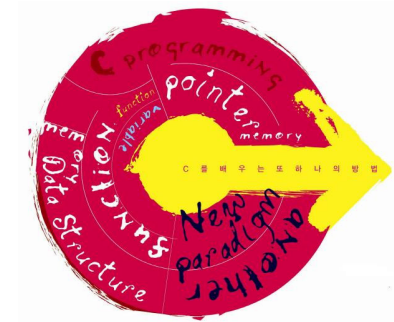
백문이불여일타(百聞而不如一打)

- 사용자 정의 함수

- 저장 공간 분류

- 함수와 포인터

- 재귀 함수



# 함수의 이해 (1/2)

## ● 함수란 무엇인가?

“함수는 필요한 데이터를 주면 정해진 행동을 하고 원하는 값을 만들어서 돌려준다”

“잘 정의된 일을 처리하는 기본 단위”

데이터만 주면,  
내가 그 모든 값의 평균을  
구해주시



나에게 식빵을 주면,  
구운 식빵을 만들어 드리죠!!



나에게 정수 2개만 주면,  
그 곱을 만들어 주겠어!!



# 함수의 이해 (2/2)

---

- 왜 함수를 사용하는가?

- 분할과 정복(Divide-and Conquer)

- 어떤 문제를 해결하기 위해 여러 개의 작은 문제로 쪼개는 것

- 프로그램의 작성이 용이

- 반복적인 일을 수행하는 경우 원시파일의 크기를 줄일 수 있다.

- (장점) 함수 단위로 프로그램 구성 시...

- **모듈화:** 함수 호출을 통한 프로그램 간략화
      - 함수 재 사용성을 통한 프로그램 구성의 편리성
    - 표준 함수 이용을 통한 프로그램 구현의 용이성
    - 함수 단위 데이터 접근 방법(변수의 지역성)을 통해 자료에 대한 제어의 용이성



# 함수의 이해

사용자 정의 함수



# 사용자 정의 함수 (1/5)

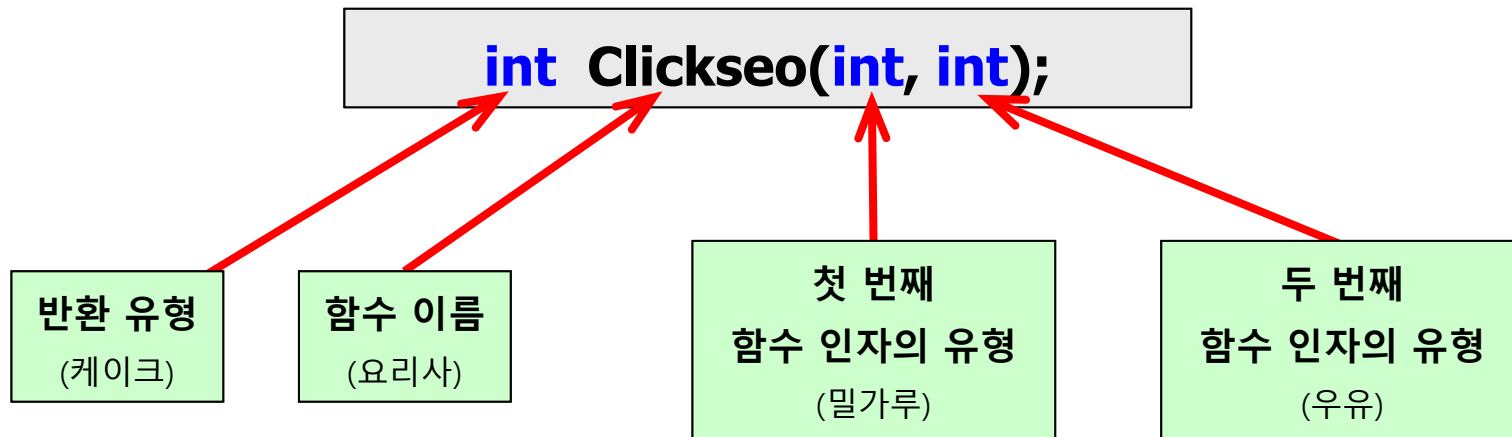
## ● 사용자 정의 함수: 함수 원형 선언 및 정의

### 1. 함수 원형 선언

- 컴파일러에게 함수의 이름과 반환 값의 유형(Return Type) 그리고 함수 인자의 유형들을 알려준다.

### 2. 함수 정의

- 함수에 대한 코드를 생성하는 작업
- 함수의 구성요소(함수 헤더, 함수 몸체)

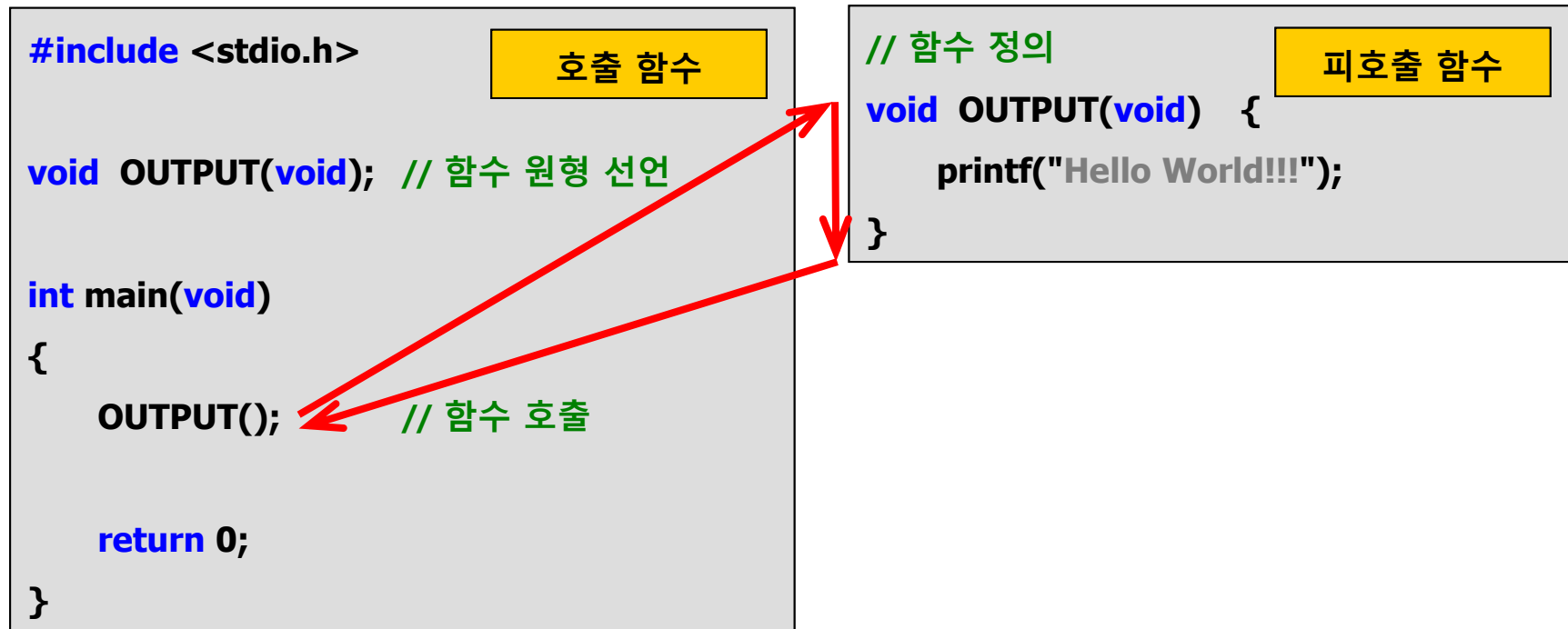


# 사용자 정의 함수 (2/5)

- 사용자 정의 함수: 함수 호출

- 3. 함수 호출

- 호출 함수와 피호출 함수





# 사용자 정의 함수 (3/5)

- 단일 함수로 프로그램 작성

```
#include <stdio.h>
int main(void)
{
    int    a, b, sum;

    scanf_s("%d %d", &a, &b);
    // scanf("%d %d", &a, &b);

    sum = a + b;

    printf("%d + %d = %d\n", a, b, sum);

    return 0;
}
```

a, b의 두 정수를 더하는 명령행을 하나의 함수로 작성

# 사용자 정의 함수 (4/5)

- 여러 개의 함수로 프로그램 작성

```
#include <stdio.h>
int ADD( int, int );

int main(void)
{
    int    a, b, sum;

    scanf_s("%d %d", &a, &b);
    // scanf("%d %d", &a, &b);

    sum = ADD(a, b);

    printf("%d + %d = %d\n", a, b, sum);

    return 0;
}
```

```
int ADD(int a, int b) {
    int    sum;
    sum = a + b;
    return sum;
}
```

# 사용자 정의 함수 (5/5)

## 예제 5-1: 사용자 정의 함수 -- ADD 함수

```
#include <stdio.h>

// 함수 원형 선언
int ADD(int, int);

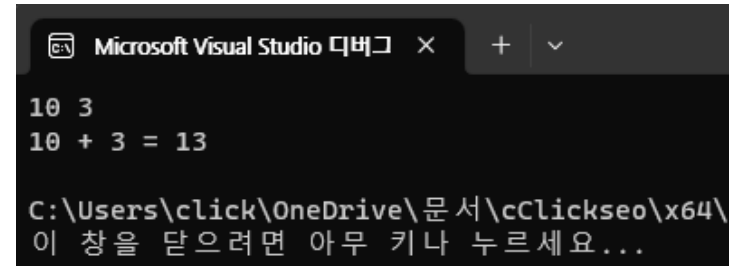
int main(void)
{
    int    a, b, sum;

    scanf_s("%d %d", &a, &b);
    // scanf("%d %d", &a, &b);

    // 함수 호출: ADD 함수
    sum = ADD(a, b);
    printf("%d + %d = %d\n", a, b, sum);

    return 0;
}

// 함수 정의: ADD 함수
int ADD(int a, int b)    {
    int    sum = a + b;
    return sum;          // return a + b;
}
```



```
Microsoft Visual Studio 디버그 x + v
10 3
10 + 3 = 13
C:\Users\click\OneDrive\문서\cClickseo\x64\
이 창을 닫으려면 아무 키나 누르세요...
```

# 저장 공간 분류



- 함수의 이해
- 저장 공간 분류
  - 지역 변수와 전역 변수
  - 자동 변수와 정적 변수
  - 외부 변수
  - 레지스터 변수

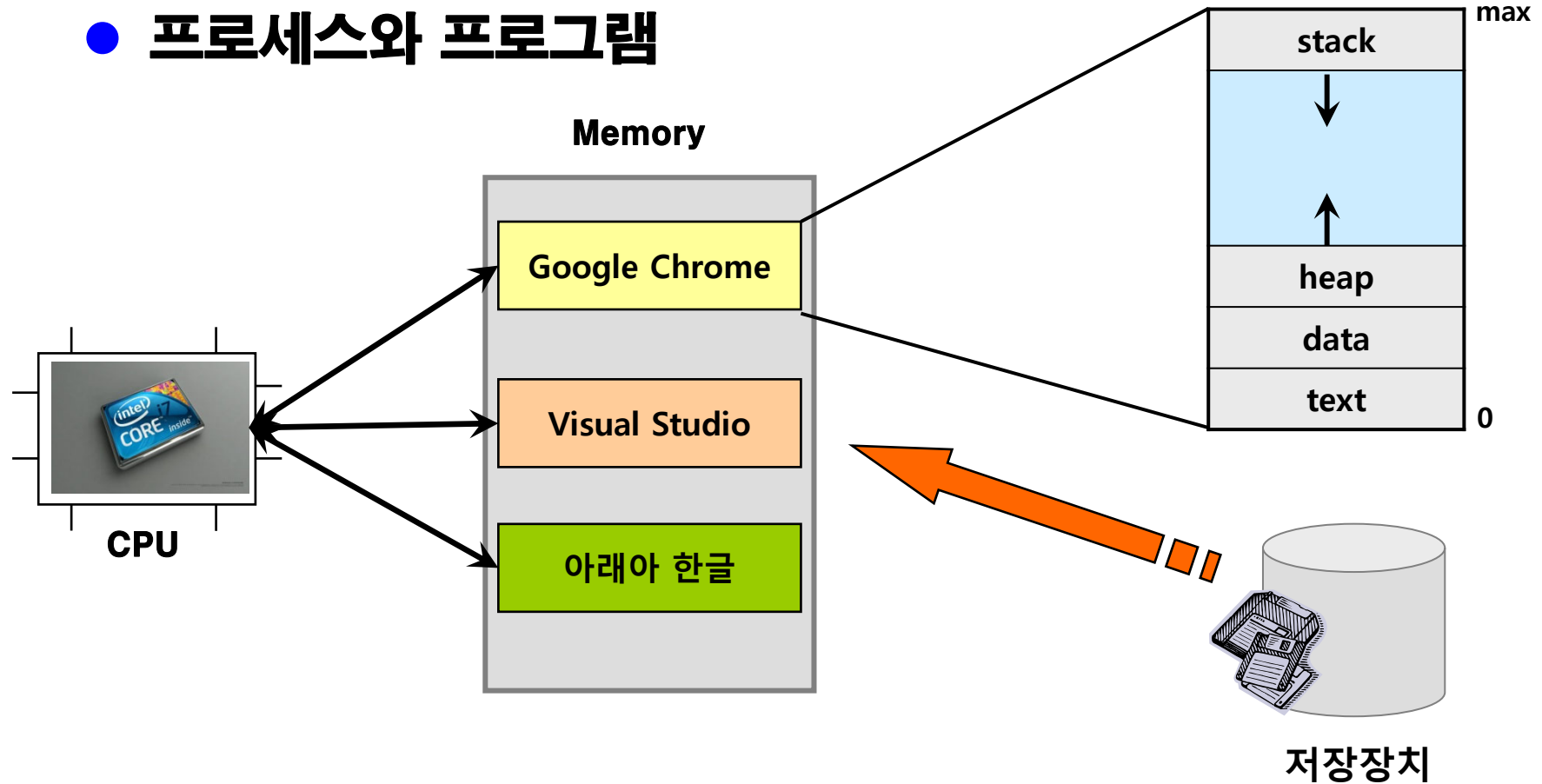
백문이불여일타(百聞而不如一打)

- 함수와 포인터
- 재귀 함수



# 저장 공간 분류 (1/2)

- 프로세스와 프로그램



**프로세스:** 운영체제에서 프로세스는 "실행중인 프로그램"

**프로그램:** 컴퓨터를 실행시키기 위해 차례대로 작성된 "명령어 집합"

# 저장 공간 분류 (2/2)

- 기억 장소 활용에 따른 변수의 종류

변수의 종류	예약어	생존 기간	유효 범위	초기화	초기화 값
자동 변수	(auto)	일시적	지역적	수행 시	임의 값
레지스터 변수	register				
(내부) 정적 변수	static	영구적	지역적	컴파일 시	0
(외부) 정적 변수			전역적		
외부 변수					



## 저장 공간 분류

지역 변수, 전역 변수



# 지역 변수와 전역 변수 (1/5)

## ● 지역 변수(Local Variable)

### ○ 함수 또는 블록 안에 정의된 변수

- 사용 범위가 함수 내부로 제한(블록 안에서만 참조 가능)
- **함수 호출 시 생성**(메모리 할당)되고, **함수 종료 시 소멸**(메모리 반납)
  - 자동변수(automatic variable)
- 서로 다른 함수에 같은 변수 이름 사용 가능
- 예: `auto`, `register`, 함수 내부에서 선언된 `static`

### ○ 지역 변수의 초기화

- 함수가 호출될 때마다 메모리를 할당 받고, 종료 시 메모리 반납
  - 초기화를 하지 않은 지역 변수는 임의의 값을 가진다.

“지역 변수는 함수가 호출될 때 메모리 상에 올라갔다가(메모리 할당),  
함수가 종료되면 메모리상에서 사라진다(메모리 반납).”



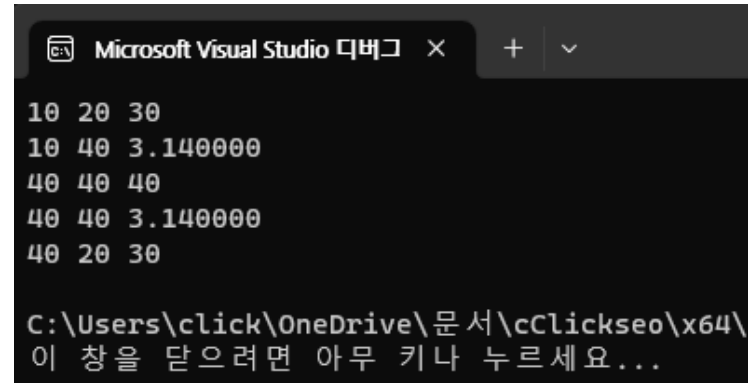
# 지역 변수와 전역 변수 (2/5)

## 예제 5-2: 지역 변수

```
#include <stdio.h>
int main(void)
{
    int    a = 10, b = 20, c = 30;

    printf("%d %d %d\n", a, b, c);
    {
        int    b = 40;
        double c = 3.14;
        printf("%d %d %f\n", a, b, c);

        a = b;
        {
            int    c;
            c = b;
            printf("%d %d %d\n", a, b, c);
        }
        printf("%d %d %f\n", a, b, c);
    }
    printf("%d %d %d\n", a, b, c);
    return 0;
}
```



```
Microsoft Visual Studio 디버그 × + ▾
10 20 30
10 40 3.140000
40 40 40
40 40 3.140000
40 20 30
C:\Users\click\OneDrive\문서\cClickseo\x64\
이 창을 닫으려면 아무 키나 누르세요...
```

# 지역 변수와 전역 변수 (3/5)

## ● 전역 변수(Global Variable)

### ○ 모든 함수가 함께 사용(공유)하는 변수

- 함수 외부에서 선언
- 프로그램 시작 시 생성(메모리 할당)되고, 프로그램 종료 시 소멸(메모리 반납)
- 같은 이름의 전역 변수는 하나 이상 사용할 수 없다.
- 예: `extern`, 함수 외부에서 선언 된 `static`

### ○ 전역 변수의 초기화

- 프로그램 시작 시 초기화되고, 프로그램 종료 시까지 값을 유지한다.
  - 초기값을 지정하지 않은 경우 0 으로 자동 초기화된다.

“전역 변수와 정적 변수의 사용은 최대한 피해야 한다”

# 지역 변수와 전역 변수 (4/5)

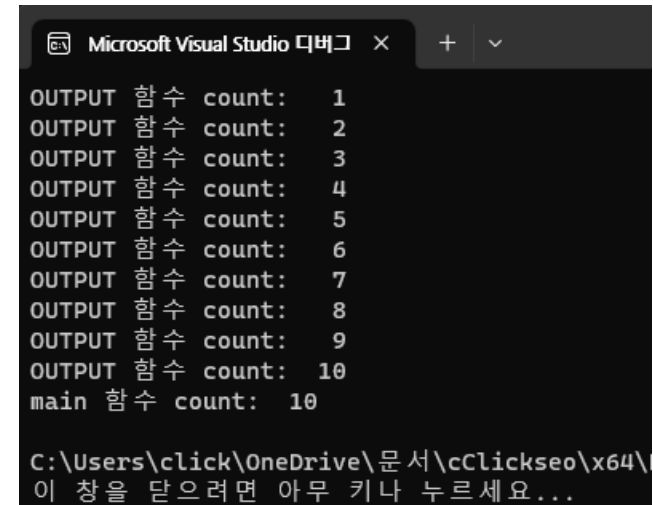
## 예제 5-3: 전역 변수

```
#include <stdio.h>
void OUTPUT(void);

// 전역변수
int count;

int main(void)
{
    for(int i=0; i<10; i++)
        OUTPUT();
    printf("main 함수 count: %3d\n", count);
    return 0;
}

void OUTPUT(void)
{
    count++;
    printf("OUTPUT 함수 count: %3d\n", count);
    return;
}
```



```
Microsoft Visual Studio 디버그 x + v
OUTPUT 함수 count: 1
OUTPUT 함수 count: 2
OUTPUT 함수 count: 3
OUTPUT 함수 count: 4
OUTPUT 함수 count: 5
OUTPUT 함수 count: 6
OUTPUT 함수 count: 7
OUTPUT 함수 count: 8
OUTPUT 함수 count: 9
OUTPUT 함수 count: 10
main 함수 count: 10
C:\Users\click\OneDrive\문서\cClickseo\x64\
이 창을 닫으려면 아무 키나 누르세요...
```

# 지역 변수와 전역 변수 (5/5)

## 예제 5-4: 동일한 이름의 전역 변수와 지역 변수

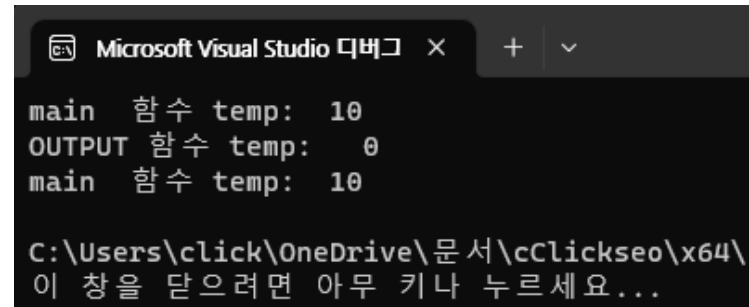
```
#include <stdio.h>
void OUTPUT(void);

// 전역변수 : temp
int temp;

int main(void)
{
    // 지역변수 : temp
    int temp = 10;

    printf("main 함수 temp: %3d\n", temp);
    OUTPUT();
    printf("main 함수 temp: %3d\n", temp);
    return 0;
}

void OUTPUT(void)
{
    printf("OUTPUT 함수 temp: %3d\n", temp);
    return;
}
```



```
Microsoft Visual Studio 디버그 x + v
main 함수 temp: 10
OUTPUT 함수 temp: 0
main 함수 temp: 10
C:\Users\click\OneDrive\문서\cClickseo\x64\
이 창을 닫으려면 아무 키나 누르세요...
```

“지역 내에서는 지역 변수가  
전역 변수보다 우선 시 된다.”



## 저장 공간 분류

자동 변수, 정적 변수



# 자동 변수와 정적 변수 (1/3)

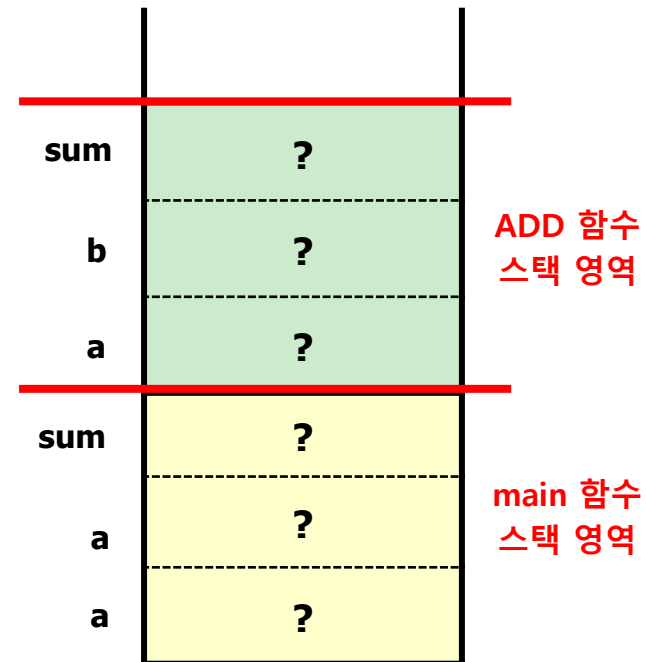
## ● 자동 변수(Auto Variable)

**auto** 자료형 변수명;

- 선언된 블록 범위 안에서 자동 생성되고 자동 반납된다.
  - 지역 변수의 또 다른 표현
    - 예: 함수 내부에서 선언되는 지역변수, 매개변수
  - 스택 사용: 선언만 하고 초기화 하지 않으면 임의의 값을 갖는다.

```
#include <stdio.h>
int ADD(int a, int b);
int main(void)
{
    auto int a=10, b=20, sum;
    sum = ADD(a, b);
    return 0;
}

int ADD(int a, int b) {
    int sum;
    sum = a + b;
    return sum;
}
```



# 자동 변수와 정적 변수 (2/3)

## ● 정적 변수(Static Variable)

`static` 자료형 변수명;

### ○ 프로그램 종료 전까지 할당 받은 메모리 공간을 유지한다.

#### • 정적 변수의 초기화

- 프로그램 시작 시 초기화되며, 프로그램 종료 시 까지 유지된다.
- 초기값을 지정하지 않을 경우 자동 초기화 된다.

### ○ 지역 정적 변수

#### • 함수 내부에서 선언되며, 해당 함수 안에서 사용 되는 지역변수

- 지역 변수의 값을 프로그램 종료 시 까지 유지하기 위해 사용
- 함수 종료 후에도 소멸(메모리 반납)되지 않으며, 다시 함수 호출 시 그 직전의 값을 참조

### ○ 전역 정적 변수

#### • 함수 외부에서 선언되며, 전역변수로 사용 되는 정적 변수

- 정적 변수를 전역 변수로 사용하는 경우 다른 파일에서 접근 불가

# 자동 변수와 정적 변수 (3/3)

## 예제 5-5: 정적 변수

```
#include <stdio.h>

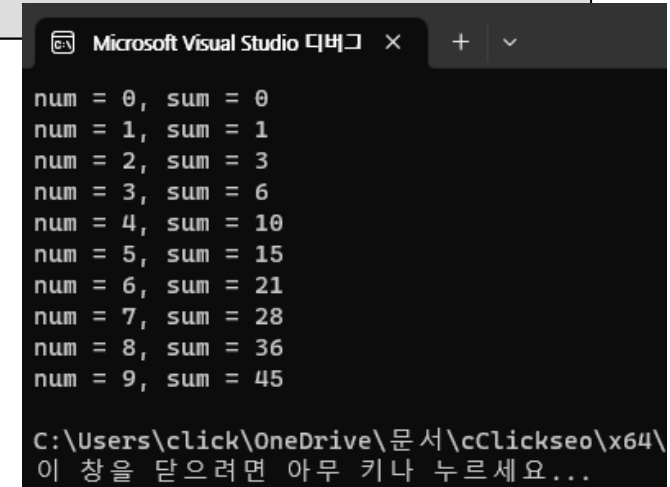
void SUM(int);

int main(void)
{
    for(int i=0; i<10; i++)
        SUM(i);

    // Error : 'sum' : undeclared identifier
    // printf("i = %d, sum = %d \n", i, sum);

    return 0;
}

void SUM(int num)
{
    // (지역)정적변수: 처음 함수 호출 시 한 번만 초기화
    static int sum = 0;
    sum += num;
    printf("num = %d, sum = %d \n", num, sum);
    return;
}
```



```
Microsoft Visual Studio 디버그 x + v
num = 0, sum = 0
num = 1, sum = 1
num = 2, sum = 3
num = 3, sum = 6
num = 4, sum = 10
num = 5, sum = 15
num = 6, sum = 21
num = 7, sum = 28
num = 8, sum = 36
num = 9, sum = 45
C:\Users\click\OneDrive\문서\cClickseo\x64\
이 창을 닫으려면 아무 키나 누르세요...
```





## 저장 공간 분류

외부 변수, 레지스터 변수



# 외부 변수

## ● 외부 변수(Extern Variable)

**extern** 자료형 변수명;

### ○ 다른 파일에 정의 된 전역변수를 선언만 하고 사용

- 별도의 기억 장소 할당 없이 기존의 변수와 기억 장소 공유
  - 변수의 영향력이 가장 넓다: **남용은 금물!!!**
- 외부 변수의 장점
  - 매개 변수처럼 전달이 불필요하며, 시간이 소비되지 않으므로 보다 효율적이다.
  - 변수들의 종류 중에서 통용 범위가 가장 넓고 생존기간이 영구적이다(모든 파일).
- 외부 변수의 단점
  - 프로그램을 크고 복잡하게 만들어 부작용의 위험성
  - 함수 인수의 주고 받음이 불분명: **함수 간의 독립성 상실**
  - 외부 변수를 남용하면 모듈 간의 자료결합이 강하게 되어 프로그램의 구조를 해치게 되므로 생산성이 저하된다: **함수 간의 강결합성**

1) 프로그램 전체를 총괄하는 변수

2) 프로그램 전체의 상황을 기억하는 변수

와 같은 경우에 사용한다(반드시 그러해야 되는 것은 아니지만...).

# 레지스터 변수 (1/2)

## ● 레지스터 변수(Register Variable)

### ○ 메모리가 아닌 CPU의 레지스터에 저장되는 변수

- 처리 속도가 빠르지만, 사용 가능한 레지스터 개수는 제한된다.
  - 보통 2,3 개 정도만 레지스터 변수로 사용 가능하다.
  - 주의: 레지스터 변수에 주소 연산자(&)는 사용할 수 없다.
- 자동변수와 기능적으로 동일하다.

register 자료형 변수명;

### ○ 사용 가능한 데이터 형 또한 char, int, 포인터 형으로 제한된다.

- 외부 변수나 정적 변수는 레지스터 변수로 정의할 수 없다.

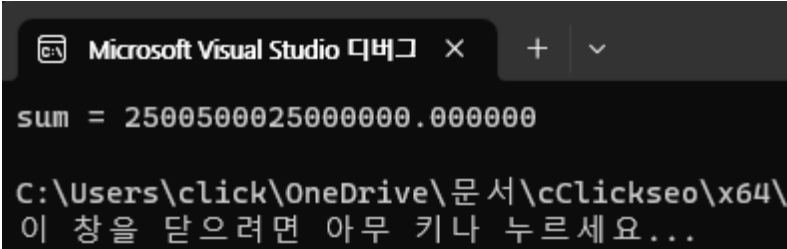
# 레지스터 변수 (2/2)

## 예제 5-6: 레지스터 변수

```
#include <stdio.h>
int main(void)
{
    double sum = 0;

    // 레지스터 변수
    register int    i, j;
    for(i=0; i<=10000; i++) {
        for(j=0; j<=10000; j++)
            sum += i * j;
    }
    printf("sum = %f \n", sum);

    return 0;
}
```



```
Microsoft Visual Studio 디버그 x + v
sum = 2500500025000000.000000
C:\Users\click\OneDrive\문서\cClickseo\x64\
이 창을 닫으려면 아무 키나 누르세요...
```

# 함수와 포인터



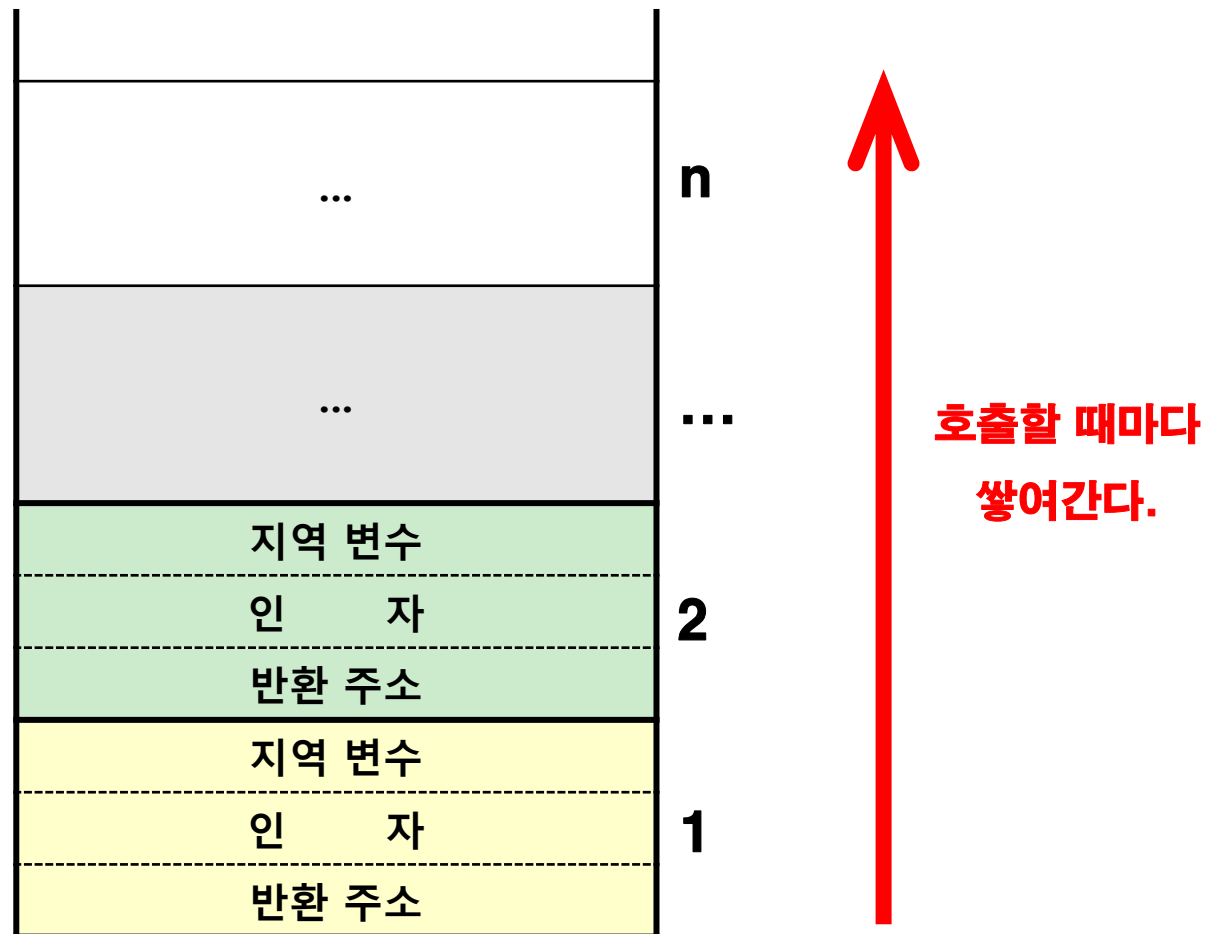
- 함수의 이해
- 저장 공간 분류
- 함수와 포인터
  - 함수 호출
  - 함수 포인터
- 재귀 함수

백문이불여일타(百聞而不如一打)



# 함수 호출 (1/4)

- 함수 호출 시 스택 상태



# 함수 호출 (2/4)

## 예제 5-7: 값에 의한 전달 -- pass by Value

```
#include <stdio.h>

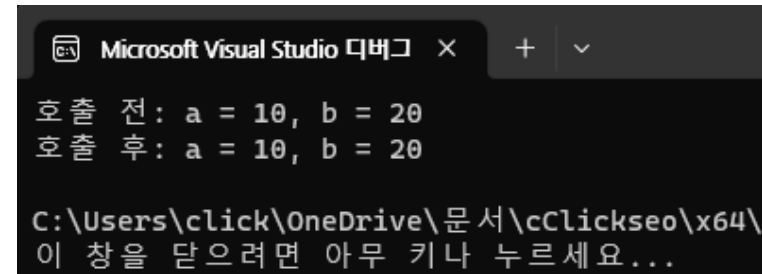
void SWAP(int, int);

int main(void)
{
    int    a = 10, b = 20;

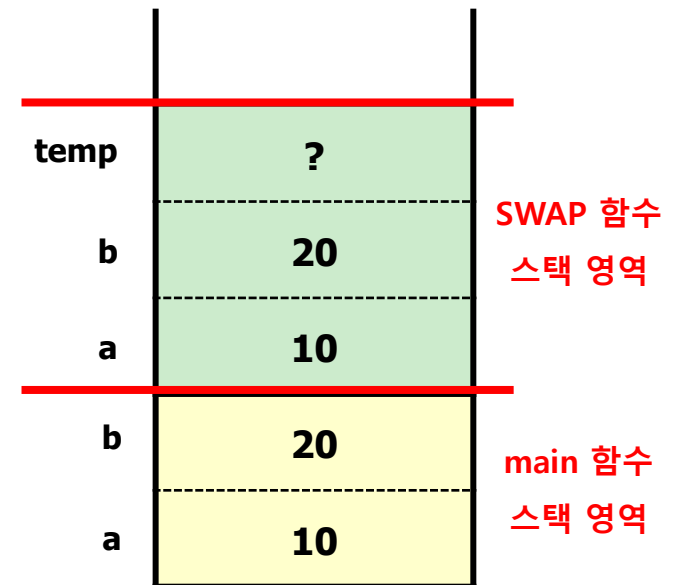
    printf("호출 전: a = %d, b = %d \n", a, b);
    SWAP(a, b);
    printf("호출 후: a = %d, b = %d \n", a, b);

    return 0;
}

void SWAP (int a, int b)    {
    int    temp;
    temp = a;
    a = b;
    b = temp;
}
```



```
Microsoft Visual Studio 디버그 x + v
호출 전: a = 10, b = 20
호출 후: a = 10, b = 20
C:\Users\click\OneDrive\문서\cClickseo\x64\
이 창을 닫으려면 아무 키나 누르세요...
```



# 함수 호출 (3/4)

## 예제 5-8: 주소에 의한 전달 -- pass by Address

```
#include <stdio.h>

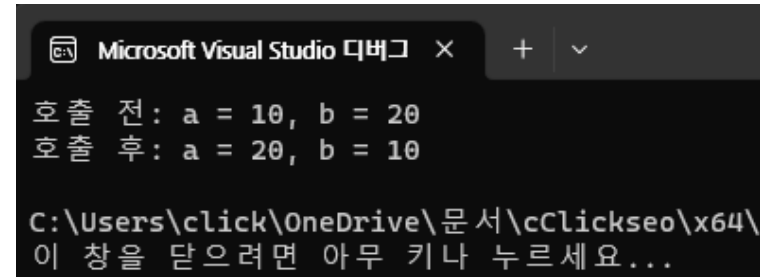
void SWAP(int*, int*);

int main(void)
{
    int    a = 10, b = 20;

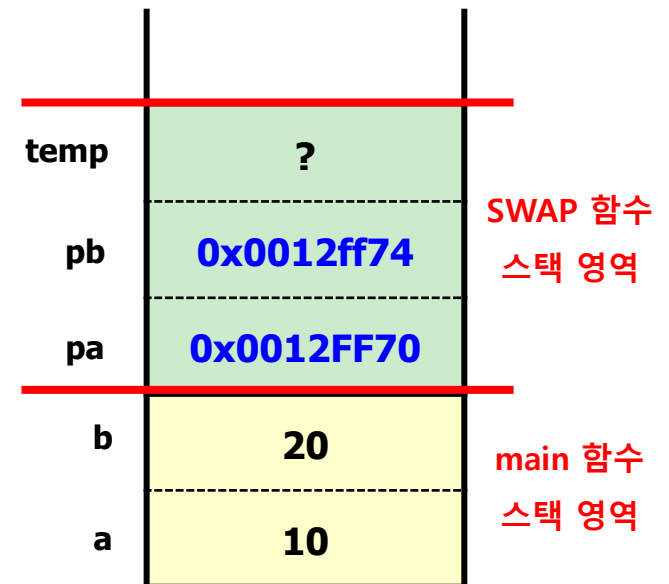
    printf("호출 전: a = %d, b = %d \n", a, b);
    SWAP(&a, &b);
    printf("호출 후: a = %d, b = %d \n", a, b);

    return 0;
}

void SWAP (int* pa, int* pb)  {
    int    temp;
    temp = *pa;
    *pa = *pb;
    *pb = temp;
}
```



```
Microsoft Visual Studio 디버그 x + v
호출 전: a = 10, b = 20
호출 후: a = 20, b = 10
C:\Users\click\OneDrive\문서\cClickseo\x64\
이 창을 닫으려면 아무 키나 누르세요...
```





# 함수 호출 (4/4)

## 예제 5-9: 포인터(메모리 주소)를 반환 값으로 갖는 함수

```
#include <stdio.h>

int*    MAX(int*, int*);

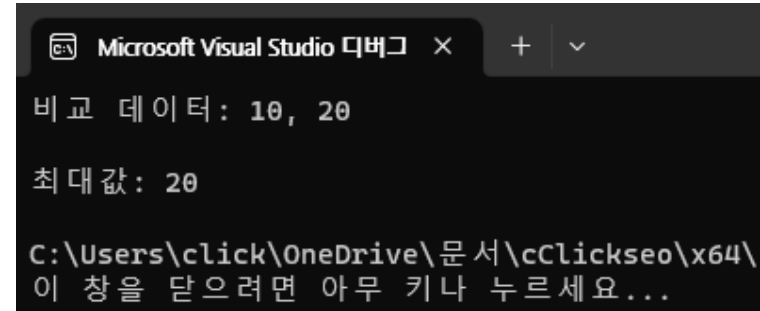
int main(void)
{
    int    a = 10, b = 20;
    int*   pMax;

    pMax = MAX(&a, &b);

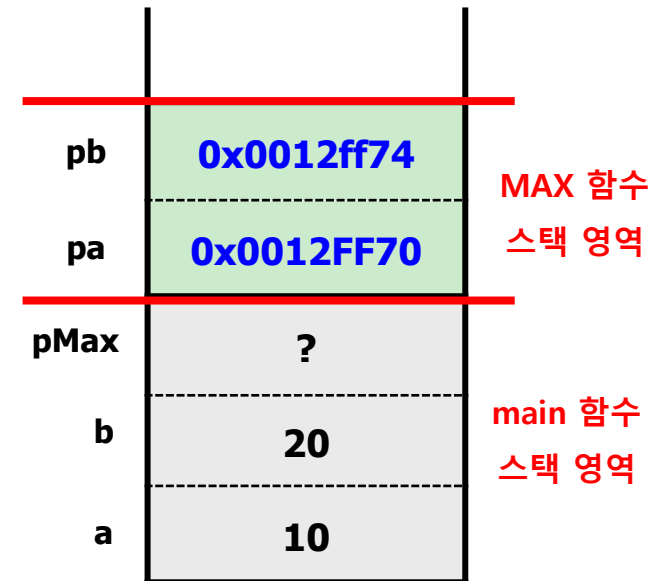
    printf("비교 데이터: %d, %d \n", a, b);
    printf("\n최대값: %d \n", *pMax);

    return 0;
}

int*    MAX(int* pa, int* pb)  {
    return  *pa > *pb ? pa : pb;
}
```



```
Microsoft Visual Studio 디버그
비교 데이터: 10, 20
최대값: 20
C:\Users\click\OneDrive\문서\cClickseo\x64\
이 창을 닫으려면 아무 키나 누르세요...
```





# 함수와 포인터

## 함수 포인터



# 함수 포인터 (1/3)

## ● 함수 포인터(Function Pointer): 간접 호출

### ○ 함수를 가리킬 수 있는 포인터

- 자주 사용되는 함수의 주소를 배열에 저장해 두고 호출하면 속도가 빨라진다.
- 잘 쓰이지는 않지만, 수치해석이나 그래픽 같은 분야에서 사용

```
#include <stdio.h>

int ADD(int, int);

int main(void)
{
    int a = 10, b = 20, sum = 0;
    int (*p)(int, int) = ADD;    // 함수 포인터 변수 선언 및 초기화

    sum = p(a, b);    // 함수 포인터를 이용한 함수 호출(간접 호출)
    printf("%d + %d = %d \n", a, b, sum);

    return 0;
}

int ADD( int a, int b ) {
    return a + b;
}
```

# 함수 포인터 (2/3)

## 예제 5-10: 함수 포인터와 사칙 연산 계산 함수

(1/2)

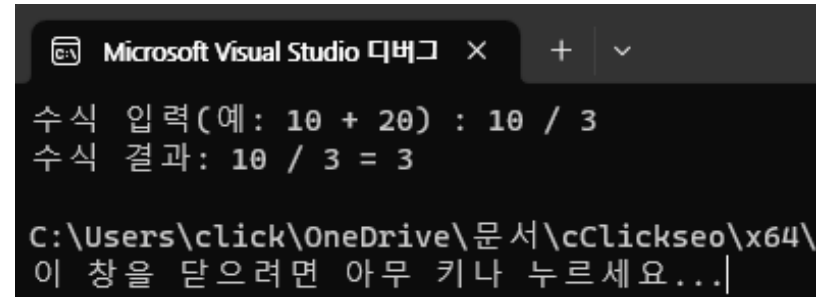
```
#include <stdio.h>
#include <stdlib.h> // exit

int  ADD(int, int);
int  SUB(int, int);
int  MUL(int, int);
int  DIV(int, int);
int  CAL(int, int, int (*)(int, int));

int main(void)
{
    char  op;
    int   a, b, res;

    printf("수식 입력(예: 10 + 20) : ");
    scanf_s("%d %c %d", &a, &op, (int)sizeof(op), &b);
    // scanf("%d %c %d", &a, &op, &b);

    switch( op ) {
        case '+': res = CAL(a, b, ADD); break;
        case '-': res = CAL(a, b, SUB); break;
        case '*': res = CAL(a, b, MUL); break;
        case '/': res = CAL(a, b, DIV); break;
        default: printf("지원되지 않는 연산자!!!\n");
                return 0;
    }
    printf("수식 결과: %d %c %d = %d \n", a, op, b, res);
    return 0;
}
```



```
Microsoft Visual Studio 디버그 x + v
수식 입력(예: 10 + 20) : 10 / 3
수식 결과: 10 / 3 = 3
C:\Users\click\OneDrive\문서\cClickseo\x64\
이 창을 닫으려면 아무 키나 누르세요...
```

# 함수 포인터 (3/3)

예제 5-10: 함수 포인터와 사칙 연산 계산 함수

(2/2)

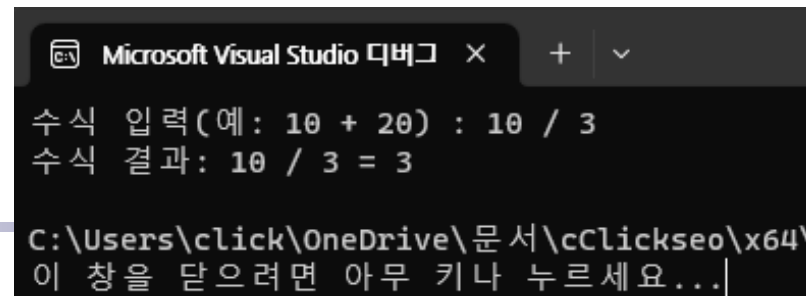
```
int CAL( int a, int b, int(*funcPtr)(int, int) ) {
    int res;

    // 함수 포인터를 이용한 간접 호출
    res = funcPtr( a, b );

    return res;
}
```

```
int ADD( int a, int b ) { return a + b; }
int SUB( int a, int b ) { return a - b; }
int MUL( int a, int b ) { return a * b; }

int DIV( int a, int b ) {
    if( b == 0 ) {
        printf("Error: 나눗셈 입력 오류!!!\n");
        printf("\t 0 으로 나눌 수 없습니다.\n");
        exit(0);
    }
    return a / b;
}
```



```
Microsoft Visual Studio 디버그
수식 입력(예: 10 + 20) : 10 / 3
수식 결과: 10 / 3 = 3
C:\Users\click\OneDrive\문서\cClickseo\x64\
이 창을 닫으려면 아무 키나 누르세요...
```

# 재귀 함수



- 함수의 이해

백문이불여일타(百聞而不如一打)

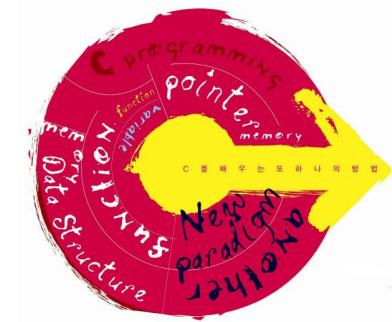
- 저장 공간 분류

- 함수와 포인터

- 재귀 함수

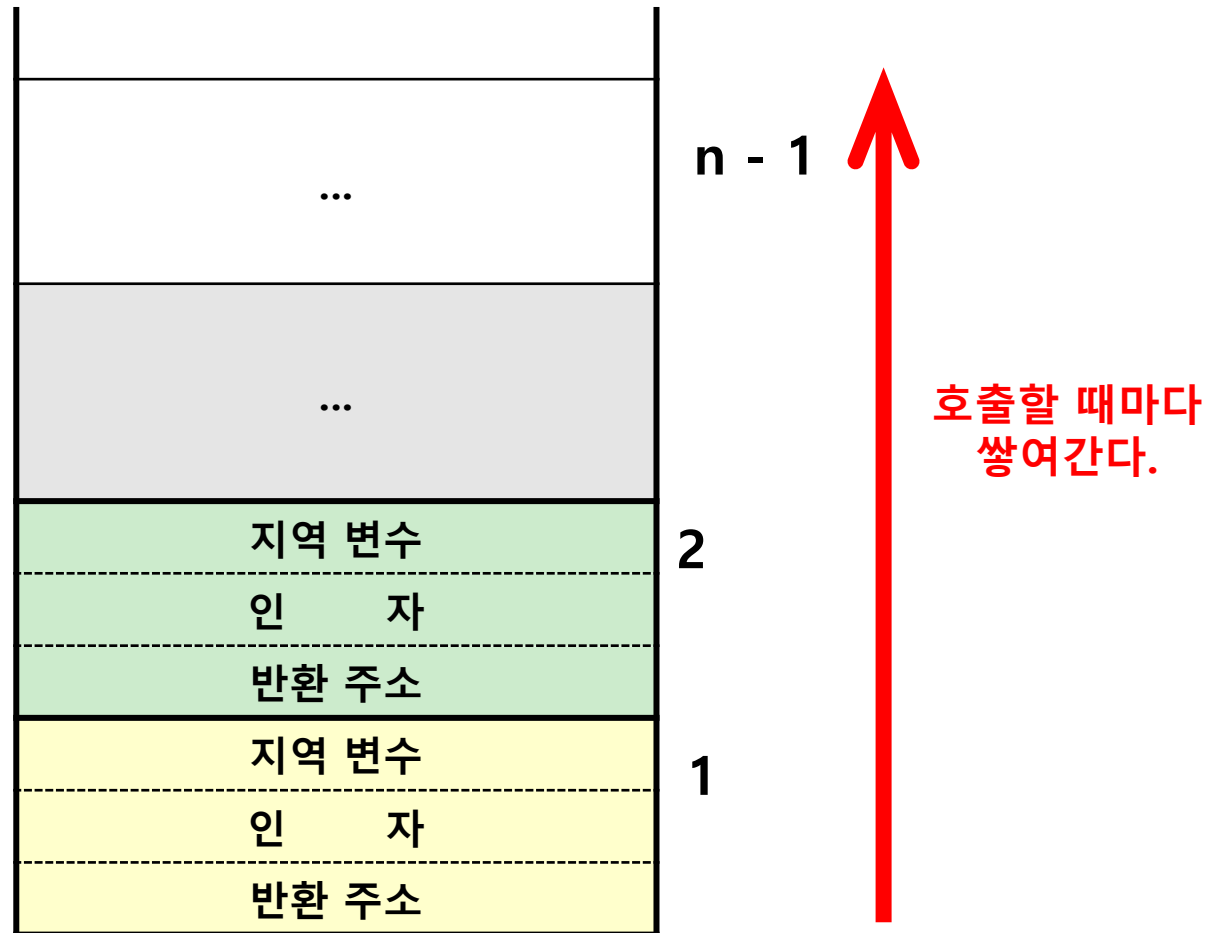
  - 반복적.재귀적 용법

  - 피보나치 수열



# 재귀 함수 (1/3)

- 재귀 함수 호출 시 스택 상태



# 재귀 함수 (2/3)

## ● 재귀 함수(Recursive Function)

### ○ 자기 자신의 함수를 호출 함으로써, 반복적인 처리를 하는 함수

- 재귀 함수 안에서 사용하는 변수는 지역변수(자동변수)
- 재귀 함수의 인수들은 값에 의한 전달(pass by Value) 방식으로 전달된다.
  
- **주의: 반드시 탈출(종료) 조건 명시!!!**
  - **Stack Overflow 오류 발생 주의!!!**
  
- **장점**
  - 코드가 훨씬 간결 해지며, 프로그램을 보기가 쉽다.
  - 또한 프로그램 오류 수정이 용이하다.
  
- **단점**
  - 코드 자체를 이해하기 어렵다.
  - 또한 메모리 공간을 많이 요구한다.



# 재귀 함수 (3/3)

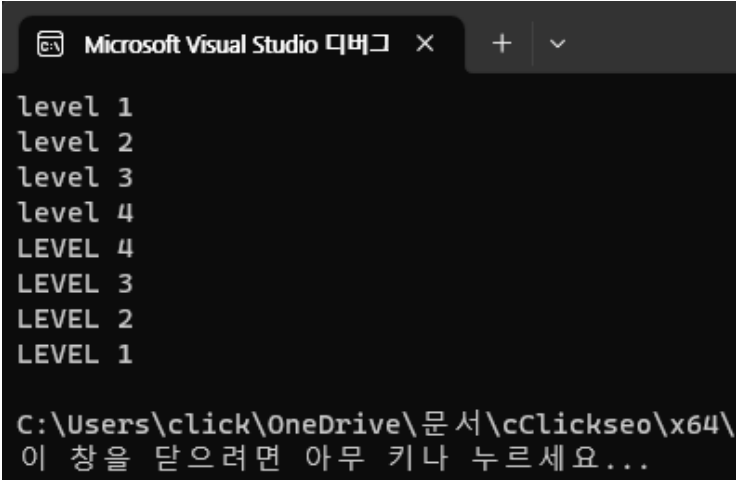
## 예제 5-11: 재귀 함수

```
#include <stdio.h>
void OUTPUT(int);

int main(void)
{
    OUTPUT(1);
    return 0;
}
```

// 재귀 함수

```
void OUTPUT(int num) {
    printf("level %d \n", num);
    if(num < 4)                // 재귀 함수 탈출(종료) 조건
        OUTPUT(num+1);
    printf("LEVEL %d \n", num);
    return;
}
```



```
Microsoft Visual Studio 디버그 x + v
level 1
level 2
level 3
level 4
LEVEL 4
LEVEL 3
LEVEL 2
LEVEL 1
C:\Users\click\OneDrive\문서\cClickseo\x64\
이 창을 닫으려면 아무 키나 누르세요...
```



# 재귀 함수

반복적.재귀적 용법



# 반복적.재귀적 용법 (1/4)

## ● 재귀적 해법

- 큰 문제에 닮은 꼴의 작은 문제가 깃든다.
- 잘 쓰면 보약, 잘못 쓰면 맹독
  - 관계중심으로 파악함으로써 문제를 간명하게 볼 수 있다.
  - 재귀적 해법을 사용하면 심한 중복 호출이 일어나는 경우가 있다.
- 재귀적 해법이 바람직한 예
  - 계승(factorial) 구하기
  - 퀵 정렬, 병합 정렬 등의 정렬 알고리즘
  - 그래프의 깊이 우선 탐색(DFS, Depth First Search)
- 재귀적 해법이 치명적인 예
  - 피보나치 수 구하기
  - 행렬 곱셈 최적순서 구하기

# 반복적.재귀적 용법 (2/4)

## ● 반복적 정의

- 반복 함수가 반복적으로 정의된다.
  - 함수 정의는 매개변수를 포함하나 함수 자체는 포함하지 않는다.

$$\text{factorial}(n) = \begin{cases} 1 & \text{if } n = 0 \\ n * (n-1) * (n-2) \dots 3 * 2 * 1 & \text{if } n > 0 \end{cases}$$

## ● 재귀적 정의

- 함수가 자기 자신을 포함한다.

$$\text{factorial}(n) = \begin{cases} 1 & \text{if } n = 0 \\ n * \text{factorial}(n-1) & \text{if } n > 0 \end{cases}$$

$\Theta(n)$



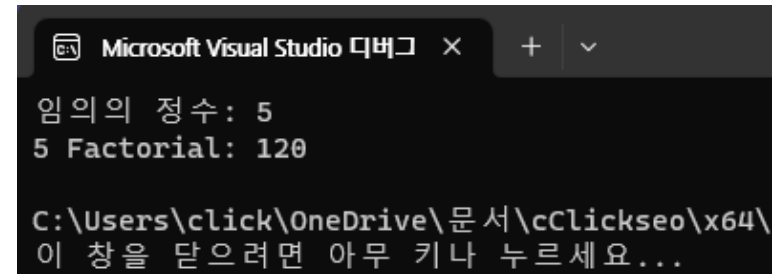
# 반복적.재귀적 용법 (3/4)

## 예제 5-12: 계승(Factorial) 구하기

```
#include <stdio.h>
int Factorial(int num);
int main(void)
{
    int    num;

    printf("임의의 정수: ");
    scanf_s("%d", &num);
    // scanf("%d", &num);

    printf("%d Factorial: %d \n", num, Factorial(num) );
    return 0;
}
```



Microsoft Visual Studio 디버그 x + v

```
임의의 정수: 5
5 Factorial: 120

C:\Users\click\OneDrive\문서\cClickseo\x64\
이 창을 닫으려면 아무 키나 누르세요...
```

```
// 재귀적 용법: Factorial
int Factorial(int num) {
    if(num == 0)           // 재귀함수 탈출(종료) 조건
        return 1;
    return num * Factorial (num - 1);
}
```

```
// 반복적 용법: Factorial
int Factorial(int num) {
    int    res = 1;
    for (int i = 1; i <= num; i++)
        res = res * i;
    return res;
}
```

# 반복적.재귀적 용법 (4/4)

## 예제 5-13: 1부터 n까지의 합

```
#include <stdio.h>
int SUM(int num);
int main(void)
{
    int    num;

    printf("임의의 정수 입력: ");
    scanf_s("%d", &num);           // scanf("%d", &num);

    printf("1부터 %d까지의 합: %d\n", num, SUM(num) );
    return 0;
}
```

```
Microsoft Visual Studio 디버그 x + v
임의의 정수 입력: 10
1부터 10까지의 합: 55

C:\Users\click\OneDrive\문서\cClickseo\x64\
이 창을 닫으려면 아무 키나 누르세요...
```

```
// 재귀적 용법: 1부터 n까지의 합
int SUM(int num) {
    if(num < 0)
        return 0;
    return num + SUM(num - 1);
}
```

## 알고리즘 분석

$$sum(n) = 1 + 2 + 3 + \dots + (n - 1) + n$$

```
// 반복적 용법: 1부터 n까지의 합
int SUM(int num) {
    int    tot = 0;
    for (int i = 1; i < num + 1; i++)
        tot += i;
    return tot;
    // return num * (num+1) / 2
}
```

# O(n)  
# O(1)



# 재귀 함수

## 피보나치 수열

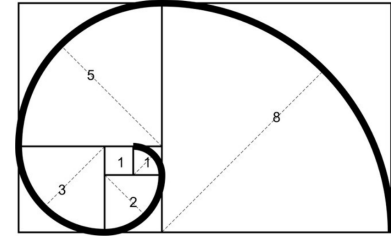


# 피보나치 수열 (1/3)

## ● 피보나치 수열(Fibonacci Sequence)

### ○ 피보나치(Fibonacci)

- 1,200년 경에 활동한 이탈리아 수학자



“토끼 한 마리가 매년 새끼 한 마리를 낳는다.  
새끼는 한 달 후부터 새끼를 낳기 시작한다.  
최초 토끼 한 마리가 있다고 하면...  
한 달 후에 토끼는 두 마리가 되고 두 달 후에는 세 마리가 되고...”

// 재귀적 용법: 피보나치 수열

**Fibonacci**(num)

{

if (num = 1 or num = 2)  
then return 1;

else

return (**Fibonacci**(num - 1) + **Fibonacci**(num - 2));

}

$$f_n = f_{n-1} + f_{n-2} (n \geq 3)$$

$$f_1 = f_2 = 1 (n = 1, 2)$$

“아주 간단한 문제지만...  
동적 프로그래밍의 동기와 구현이 다 포함되어 있다.”



# 피보나치 수열 (2/3)

## 연습문제 5-14: 피보나치 수열 -- 재귀적 용법

```
#include <stdio.h>
int Fibo(int num);

int main(void)
{
    int    num;

    printf("### 피보나치 수열 구하기 ### \n\n");
    printf("몇 번째 수열까지 출력할까요: ");
    scanf_s("%d", &num); // scanf ("%d", &num);

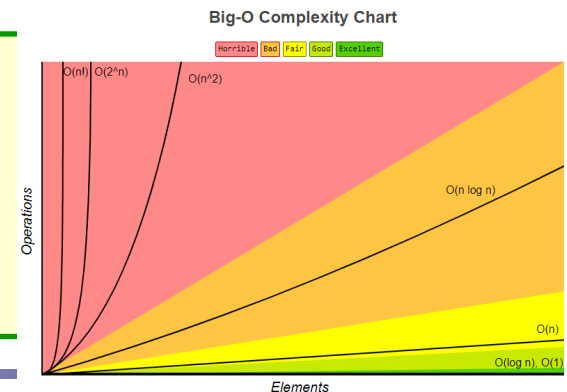
    for( int i=1; i<=num; i++ ) {
        if(i%5)          printf("%8d", Fibo(i));
        else             printf("%8d\n", Fibo(i));
    }
    printf("\n");
    return 0;
}
```

```
// 재귀적 용법: 피보나치 수열
int Fibo(int num) {
    // 재귀 호출: 탈출 조건
    if(num == 1 || num == 2)
        return 1;
    return Fibo(num - 1) + Fibo(num - 2); // O(2^n)
}
```

```
Microsoft Visual Studio 디버그 x + v
### 피보나치 수열 구하기 ###

몇 번째 수열까지 출력할까요: 35
   1      1      2      3      5
   8     13     21     34     55
  89    144    233    377    610
 987   1597   2584   4181   6765
10946 17711 28657 46368 75025
121393 196418 317811 514229 832040
1346269 2178309 3524578 5702887 9227465

C:\Users\click\OneDrive\문서\cClickseo\x64\
이 창을 닫으려면 아무 키나 누르세요...
```

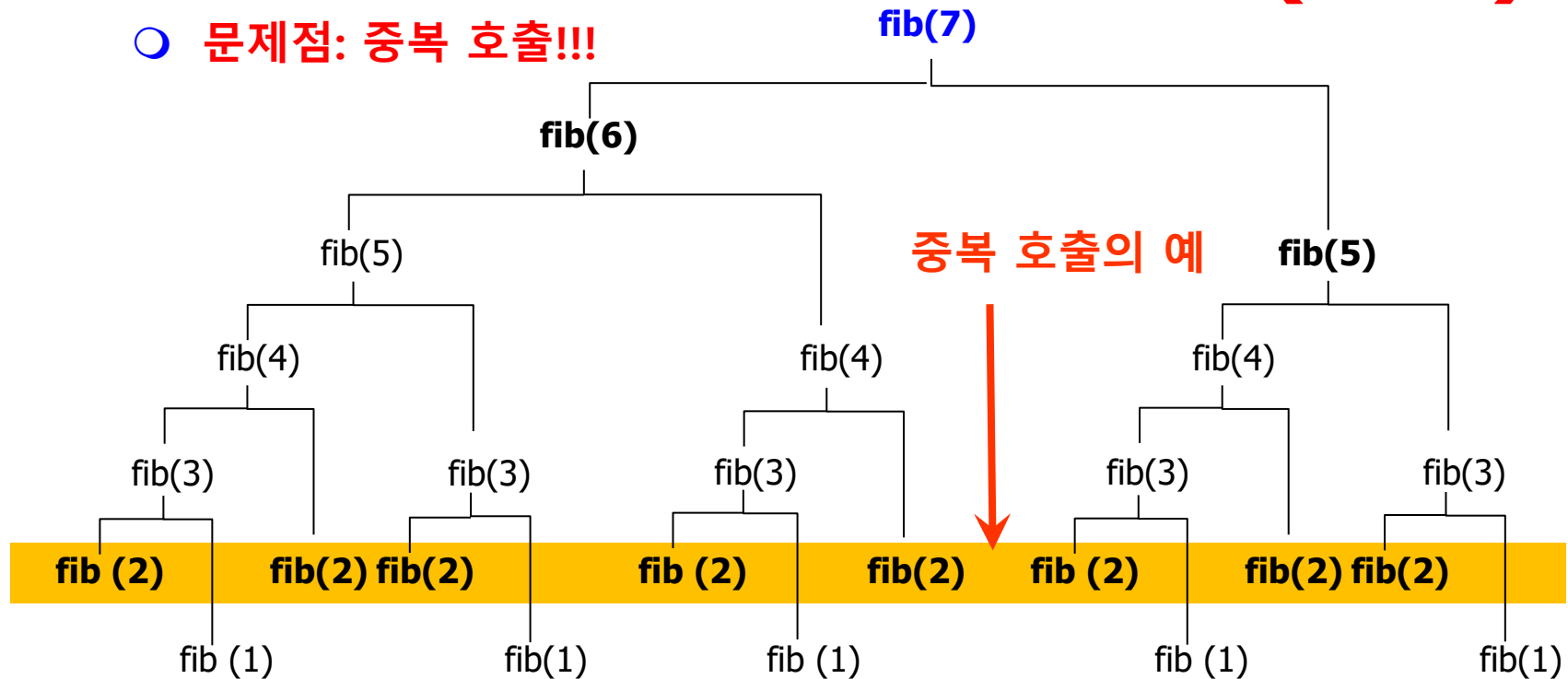


# 피보나치 수열 (3/3)

- 피보나치 수열: 재귀적 용법

$O(2^n)$

- 문제점: 중복 호출!!!



“엄청난 중복 호출이 존재한다.”

--> 재귀적 알고리즘은 **지수함수에 비례**하는 시간이 든다.



# 재귀 함수

피보나치 수열: 동적 프로그래밍



# 피보나치 수열: 동적 프로그래밍 (1/2)

## ● 동적 프로그래밍의 적용 조건

### ○ 최적 부분 구조(Optimal Substructure)

- 큰 문제의 해답에 그보다 작은 문제의 해답이 포함되어 있다.
  - 최적 부분 구조를 가진 문제의 경우에는 재귀 호출을 이용하여 문제를 풀 수 있다.

### ○ 재귀 호출 시 중복(overlapping recursive calls)

- 재귀적으로 구현했을 때 중복 호출로 심각한 비효율이 발생한다.

동적 프로그래밍이 그 해결책 !!!

- 위의 두 성질이 있는 문제에 대해 적절한 저장 방법으로 중복 호출의 비효율을 제거한 것을 동적 프로그래밍이라고 한다.

# 피보나치 수열: 동적 프로그래밍 (2/2)

- 피보나치 수열: 동적 프로그래밍 알고리즘

**Fibonacci(n)**

```
{  
  f[1] ← f[2] ← 1;  
  for i ← 3 to n  
    f[i] ← f[i - 1] + f[i - 2];  
  
  return f[n];  
}
```

**fibonacci**

1	1	?	?	?	?	?
---	---	---	---	---	---	---

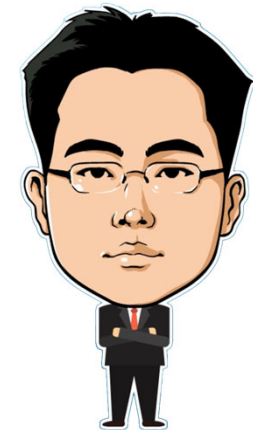
**Fibonacci(n)**

```
{  
  first ← second ← 1;  
  for i ← 3 to n  
    res ← first + second;  
  return res;  
}
```

$\Theta(n)$

# 참고문헌

- [1] 서현우, "혼자 공부하는 C 언어 : 1:1 과외 하듯 배우는 프로그래밍 자습서", 한빛미디어, 2023.
- [2] Paul Deitel, Harvey Deitel, "C How to Program", Global Edition, 8/E, Pearson, 2016.
- [3] Kamran Amini, 박지윤 번역, "전문가를 위한 C : 동시성, OOP부터 최신 C, 고급 기능까지!", 한빛미디어, 2022.
- [4] 서두옥, "(열혈강의) 또 하나의 C : 프로그래밍은 셀프입니다", 프리렉, 2012.
- [5] Behrouz A. Forouzan, Richard F. Gilberg, 김진 외 7인 공역, "구조적 프로그래밍 기법을 위한 C", 도서출판 인터비전, 2004.
- [6] Brian W. Kernighan, Dennis M. Ritchie, 김석환 외 2인 공역, "The C Programming Language", 2/E, 대영사, 2004.
- [7] "C reference", cppreference.com, 2023 of viewing the site, <https://en.cppreference.com/w/c>.



이 강의자료는 저작권법에 따라 보호받는 저작물이므로 무단 전제와 무단 복제를 금지하며, 내용의 전부 또는 일부를 이용하려면 반드시 저작권자의 서면 동의를 받아야 합니다.

Copyright © Clickseo.com. All rights reserved.